i2dps daemon

`db2dps` is a small daemon running on the database server which convert rules to BGP announcements. The daemon is controlled by `systemd`. The installation is done with `make`.

The current version of `i2dps` is written in Perl. It requires the following Perl modules to be installed:

```
sudo apt-get -y install libnet-openssh-compat-perl liblist-moreutils-per
                        libnet-openssh-compat-perl libnet-ssh2-perl
                        libproc-daemon-perl libnetaddr-ip-perl
                        libdbi-perl libdbd-pg-perl libtypes-path-tiny-pe
```

Installation

On the database host, execute

```
mkdir -p /opt/db2dps && chown sysadm:sysadm /opt/db2dps
```

Edit `Makefile` and copy the source for `db2dps` to `/opt/db2dps`. You only need to change the lines to whatever your heart desire:

```
TARGETHOST      = sysadm@ddps.deic.dk
GID             = sysadm
UID             = sysadm
```

Change `TARGETHOST` and set up `ssh` credentials first. Either (depending on your local environment) do

```
./remote.sh -v make dirs
```

or copy the source to `/opt/db2dps/src` and execute:

```
cd /opt/db2dps/src && make dirs
```

If that goes well then execute

```
./remote.sh -v make all
```

or

```
cd /opt/db2dps/src && make all
```

For the C version, the target will

- fetch, extract and compile required libraries from github
- compile db2dps and place binaries etc. below `/opt/db2dps`
- install db2dps as a systemd service which will start as part of the boot process

For the Perl version the target will

- add version information to `db2dps`
- install db2dps as a systemd service which will start as part of the boot process

Usage and pseudo code below:

Name db2dps

Database / rule manipulation for DDPS

Synopsis

`db2dps [-V] [-v] [-d] [-s seconds]`

Description

`db2dps` process new *rulefiles*, and maintain rules in the database wile sending BGP flowspec updates to a number of BGP hosts. `db2dps` runs as a daemon controlled by systemd.

Options

- **-V**: print version information and exit
- **-v**: verbose and run in foreground
- **-d**: demonise
- **-s** *seconds*: sleep time between database scan. Default is 20 seconds

Pseudo code

```
read configuration || fail
check args: print version and exit | demonise | run in foreground

connect to database || exit fail

query(all my networks)

while true; do
```

```
{
  if [ exit required ]
  {
    break loop
    close database connection
    exit normal
  }
  else
  {
    sleep except seconds on first loop
  }

   if [ exist (new files with rules from fastnetmon) ]
     if (query(insert rules in database) == OK)
        delete(rulefile) or warn

  foreach bgphost do
  {
    mkrulebase("announce", bgphost)
    {
       if (bgphost requires all rules)
          query(all rules)
       else
          query(NOT isactivated and NOT expired records)
       continue if (query empty)
       {
         if (destination is within all my networks)
         {
           build rules suitable for bgphost
           send rulebase to bgp host || warn
           /* notice: this may block */
         }
         else
         {
           warn about attempt to filter for external network
         }
     }
    }
  }
  query(set isactivated for all announced rules in database)

  foreach bgphost do
  {
    mkrulebase("withdraw", bgphost)
    {
       query(all isactivated rules)
       select rules which are expired AND does not match a non-expired r
       foreach (bgphosts)
       {
         if (destination is within all my networks)
         {
```

```
          `
            build rules suitable for bgphost
            send rulebase to bgp host || warn
            /* notice: this may block */
          }
          else
          {
            warn about attempt to filter for external network
          }
        }
      }
    }
    query(set isexpired for withdrawn rules in database)
}

close database connection and exit normal
```

Author

Niels Thomas Haugård, niels.thomas.haugaard@i2.dk

Bugs

Probably. Please report them to the the author or the DDPS group. Please notice this is early work.

Rulefiles

Rulefiles has the following format, with a *header* describing the *rule type* where only `fnm` for fastnetmon is in use, rule format if we should ever change it and the *attack type* for later optimisation. The last line is literally *last-line* to avoid processing incomplete files:

```
ruleheader
rule
rule
last-line
```

The format is

```
Rule header: type;vesion;attack_info;
type: fnm     | ...
version: 1    | ...
attack_info: icmp_flood | syn_flood | udp_flood | unknown | ...
Rules: customernetworkid,uuid,fastnetmoninstanceid,administratorid,block
customernetworkid:      Customer id (int)
uuid:                   Mac address -- identify fastnetmon instance
fastnetmoninstanceid:   Customers fastnetmon # (int)
administratorid:        Administrator id (int)
blocktime:              Minutes
Type 1 - Destination Prefix
Type 2 - Source Prefix
Type 3 - IP Protocol
Type 4 — Source or Destination Port
Type 5 — Destination Port
Type 6 - Source Port
Type 7 — ICMP Type
Type 8 — ICMP Code
Type 9 - TCP flags
Type 10 - Packet length
Type 11 — DSCP
Type 12 - Fragment Encoding
```

Example:

```
head;fnm;1;syn_flood
0;00:25:90:47:2b:48;1;42;10;130.226.136.242;66.141.26.81;tcp;14372;80;80
0;00:25:90:47:2b:48;1;42;10;130.226.136.242;161.185.77.224;tcp;14374;80;
last-line
```

Some fields are read by `fnm2db` from its configuration file. The configuration file is written based on information from the database:

Var	Size	Description
customernetworkid	`int`	describing the customer
fastnetmoninstanceid	`int`	describing the customers fastnetmon which triggered the rule
administratorid	`int`	describing the (pseudo) administrator which created the rule. The administrator cannot log in, but the database requires all rule to be made by someone.

The design opens up for other kind of rule creators, e.g. Cisco Netflow which is evaluated by

CERT.

Rule creation

Just my random thoughts, but having to implement something I wonder what is the *best practice for creating rules to mitigate volumetric attacks based on flowspec*?

According to awsstatic.com DDoS attacks are most common at layers 3, 4, 6, and 7 of the Open Systems Interconnection (OSI) model.

Layer 3 and 4 attacks correspond to the Network and Transport layers of the OSI model: these are volumetric infrastructure layer attacks.

Layer 6 and 7 attacks correspond to the Presentation and Application layers of the OSI model, these are as application layer attacks and only the volumetric attacks can be detected by fastnetmon.

#	Layer	Unit	Description	Vector Examples
7	Application	Data	Network process to application	HTTP floods, DNS query floods
6	Presentation	Data	Data representation and encryption	SSL abuse
5	Session	Data	Interhost communication	N/A
4	Transport	Segments	End-to-end connections and reliability	SYN floods
3	Network	Packets	Path determination and logical addressing	UDP reflection attacks
2	Data Link	Frames	Physical addressing	N/A
1	Physical	Bits	Media, signal, and binary transmission	N/A

From awsstatic.com

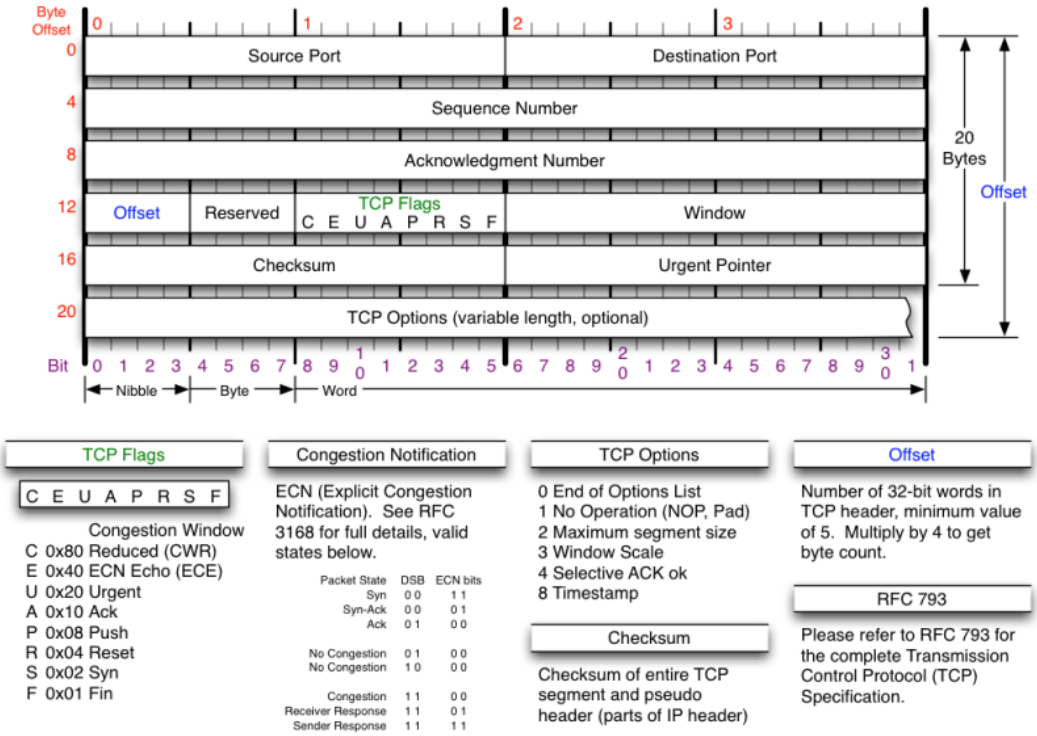
Fastnetmon detects the following type of attacks:

1. *syn_flood*: TCP packets with enabled SYN flag
2. *udp_flood*: flood with UDP packets (so recently in result of amplification)
3. *icmp flood*: flood with ICMP packets
4. *ip_fragmentation_flood*: IP packets with MF flag set or with non zero fragment offset
5. *DNS amplification*:
6. *NTP amplification*:
7. *SSDP amplification*:

8. *SNMP amplification*:

First: it is sometimes possible to distinguish between legitimate and illegitimate packets, as Not All SYNs Are Created Equal . And empty UDP and TCP packet might be rare:

For ethernet is the *minimum payload* 42 octets when an 802.1Q tag is present and 46 octets when absent according to wikipedia on ethernet frames . The minimum Layer 2 Ethernet frame size is 64 bytes for an *empty tcp or udp packet*.



We have the following values for creating a filter:

```
Type 1 - Destination Prefix
Type 2 - Source Prefix
Type 3 - IP Protocol
Type 4 — Source or Destination Port
Type 5 — Destination Port
Type 6 - Source Port
Type 7 — ICMP Type
Type 8 — ICMP Code
Type 9 - TCP flags
Type 10 - Packet length
Type 11 — DSCP
Type 12 - Fragment Encoding
```

Suggestion for rule creation:

Attack type	Mitigation	Match on
		tcp option (syn) protocol, destination port, tcp

Attack type	Mitigation	Match on
syn_flood	rate-limit	flags, size, (ttl would be nice but is still in draft), and source any
udp_flood	rate-limit	protocol and destination host and port
icmp flood	discard	protocol and destination
ip_fragmentation_flood	rate-limit	protocol and destination
DNS amplification	rate-limit	protocol, port and destination
NTP amplification	rate-limit	protocol, port and destination
SSDP amplification	discard	protocol, port 1900, source any
SNMP amplification	discard	protocol, port, destination

Note: SSDP - *Simple Service Discovery Protocol* (see draft-cai-ssdp-v1–03 does not belong on a WAN an anyway? It's used for UPnP discovery. The same goes for TCP / UDP port 1 - 19.

SNMP does to my best understanding not pass the boundaries of a company network, even not protocol version 3. And sacrificing monitoring data for the sake of the network is fine with me.

Other versions

A version of `i2dps` written in C is also available, but *currently with unresolved memory / heap errors*. It also lacks code for *white listing* and *solving the problem with overlapping rules*.

The C development environment including memory leak test with valgrind may be installed this way:

```
sudo apt-get -y update
sudo apt-get -y upgrade
sudo apt-get -y install build-essential
sudo apt-get -y install valgrind
```

Installation of the C version is documented in the `Makefile`.